15437-0542/P6541 *Patent*

UNITED STATES PATENT APPLICATION

FOR

MECHANISM FOR AUTOMATICALLY GENERATING A TRANSFORMATION DOCUMENT

INVENTOR(S):

MATTHEW D. BIRDER

PREPARED BY:
HICKMAN PALERMO TRUONG & BECKER, LLP
1600 WILLOW STREET
SAN JOSE, CALIFORNIA 95125-5106
(408) 414-1080

EXPRESS MAIL CERTIFICATE OF MAILING

"Express Mail" mailing label number EL734779814US

Date of Deposit August 16, 2001

I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Box Patent Application, Commissioner of Patents, Washington, D.C. 20231.

Tirena Say
(Typed or printed name of person mailing paper or fee)

Tirena Say
(Signature of person mailing paper or fee)

MECHANISM FOR AUTOMATICALLY GENERATING A TRANSFORMATION DOCUMENT

Inventor(s): Matthew D. Birder

Field of the Invention

This invention relates generally to computer systems, and more particularly to a mechanism for automatically generating a transformation document.

Background

The XML (eXtensible Markup Language) specification established by the W3C Organization provides a standardized methodology for exchanging structured data between different mechanisms. The different mechanisms may be different components within the same system (e.g. different program components) or they may be completely separate systems (e.g. systems of different companies, or different servers on the World Wide Web). Basically, XML allows structured data to be exchanged in a textual format using "element tags" to specify structure and to delimit different sets of data.

An example of a portion of an XML document is shown in Fig. 1. In this example, information about a person is being exchanged. To indicate that the information pertains to a person, the "person" element tags are used to delimit the data. Nested within the "person" element tags are two sets of information: (1) a name; and (2) an address. These sets of information are also delimited using the "Name" and "Address" element tags, respectively. Nested within the "Name" element tags are three child elements, namely, a first, middle, and last name, each of which is delimited by respective

element tags, and each of which has an associated value. Likewise, nested within the "Address" element tags are four child elements, namely, a street, city, state, and zip code, each of which is delimited by respective element tags, and each of which has an associated value. By delimiting the sets of data using nested element tags in this manner, the XML document makes it clear how the data is structured, and what each set of data represents. As a result, any mechanism that is capable of understanding the element tags used to delimit the data will be able to interpret and process the data. In this manner, XML makes it possible to exchange structured data in a textual, program-independent, and platform-independent manner. It is this general nature of XML that makes it so flexible and versatile. Because of its versatility, XML has grown significantly in popularity in recent years. The above discussion provides just a brief description of the XML specification. More information on XML may be found on the W3C website at www.w3c.org. All of the information on that website, as of the filing date of the present application, is incorporated herein by reference.

In some instances, before data in an XML document can be processed or rendered, the XML document first needs to be transformed. For example, if the information of the person shown in Fig. 1 is to be rendered on a cellular phone display, and the cellular phone display does not have enough room for a middle name, then the XML document may first need to be transformed by removing the "middle" name element before the information is provided to the cellular phone to be displayed. As another example, the element tag used in one system may differ from the element tag used in another system. For example, the "person" element tag in one system may correspond to the "employee" element tag in another system. Before the XML document is processed into the other

system, the XML document is first transformed to change the "person" element tag to an "employee" element tag. These are examples of simple transformations that can be made to an XML document. Many other more complex transformations may also be made.

To enable an XML document (referred to as a source document) to be transformed into another document (referred to as a target document), there is currently provided a transformation language, known as XSLT (eXtensible stylesheet language transformation). Using XSLT, a transformation document can be created which, when processed together with the source document, gives rise to the target document. In effect, the transformation document specifies the transformations that need to be made to the source document to derive the target document. For example, the transformation document may specify that whenever a "person" element tag is encountered in the source document, an "employee" element tag should be created in the target document. According to the XSLT specification, the transformation document is itself an XML document; thus, it conforms to all of the requirements to which all XML documents conform.

If it is known from the outset how a source document is to be transformed to derive a target document, then the creation of a transformation document is relatively straightforward. A user or programmer simply creates templates in the transformation document, using XSLT, to implement all of the desired transformations. In many implementations, however, it is not known how a source document is to be transformed to derive a target document. Instead, a user/programmer is simply given a source document and a target document, and asked to create a transformation document that will transform the source document into the target document. This can be a very daunting task

because it can potentially require the user/programmer to intensely analyze and compare both documents to determine the transformations that need to be made. If the two documents are lengthy, the amount of manhours required to create the transformation document could be immense. Given the difficulty and the amount of resources currently required to manually create a transformation document from a source and a target document, it is evident that a mechanism for facilitating the document creation process is needed.

Summary of the Invention

In light of the shortcomings of the prior art, there is provided, in one embodiment of the present invention, a mechanism for automatically generating a transformation document given a source document and a target document. In one embodiment, a transformation document generation mechanism (TDGM) analyzes each document to determine the structural patterns found in each. As each document is analyzed, a pattern dictionary is built that records each pattern found in each document. After the analysis of the documents is performed, the TDGM processes the pattern dictionaries to automatically generate the transformation document.

In one embodiment, for each particular pattern in the target document's pattern dictionary, the TDGM automatically generates a template in the transformation document. This template will cause the particular pattern to be created in a result document, and will be triggered when a triggering pattern is encountered in the source document. The triggering pattern is specified and associated with the template so that unless the triggering pattern is found in the source document when the source document

is processed with the transformation document, the template will not be invoked. Since it is difficult for the TDGM to determine, without purely guessing, what triggering pattern in the source document should cause the particular pattern to be created in the result document, the TDGM in one embodiment does not specify an actual triggering pattern but rather sets the triggering pattern to "iis-pattern-needed". That way, when a user reviews the transformation document after it has been generated by the TDGM, the user will know from the "iis-pattern-needed" indication that the user needs to provide a triggering pattern for the template. In one embodiment, the TDGM generates such a template in the transformation document for each particular pattern found in the target document's pattern dictionary.

In addition to the pattern creation templates noted above, the TDGM in one embodiment further generates zero or more copy templates in the transformation document. The copy templates copy identical elements (elements having the same structural format and the same data values), if any, from the source document to the result document. Once that is done, the TDGM will have generated a transformation document that can be processed with the source document to derive a result document that is at least an approximation of the target document. This transformation document may be further refined/changed by a user, but it at least provides a starting document from which the user can work. By performing much of the underlying document analysis for the user, and by generating an initial transformation document, the TDGM significantly reduces the amount of effort required on the part of the user. Thus, the TDGM greatly facilitates the transformation document creation process.

Brief Description of the Drawings

Fig. 1 illustrates a portion of a sample XML document.

Fig. 2 is a functional block diagram of a system in which one embodiment of the present invention may be implemented.

Fig. 3 shows a sample source document for use in illustrating the operation of one embodiment of the TDGM.

Fig. 4 shows a sample target document for use in illustrating the operation of one embodiment of the TDGM.

Fig. 5 is an operational flow diagram illustrating the operation of one embodiment of the TDGM.

Fig. 6 shows a tree representation of the sample source document of Fig. 3.

Fig. 7 shows a tree representation of the sample target document of Fig. 4.

Fig. 8 shows a pattern dictionary for the sample source document of Fig. 3 generated in accordance with one embodiment of the present invention.

Fig. 9 shows a pattern dictionary for the sample target document of Fig. 4 generated in accordance with one embodiment of the present invention.

Figs. 10A-10D show a sample transformation document generated in accordance with one embodiment of the present invention based upon the sample documents of Figs. 3 and 4.

Fig. 11 is a hardware block diagram of a computer system in which one embodiment of the present invention may be implemented.

Detailed Description of Embodiment(s)

FUNCTIONAL OVERVIEW

With reference to Fig. 2, there is shown a functional block diagram of a system 200 in which one embodiment of the present invention may be implemented. As shown, system 200 comprises a user interface (UI) 202, a transformation document generation mechanism (TDGM) 204, and a transformation processor (TP) 206. The UI 202 provides a mechanism for enabling a user to interact with the other components 204, 206 of the system 200. For purposes of the present invention, UI 202 may be any type of user interface, including but not limited to a simple text-based user interface and a full-function graphical user interface. UI 202 enables a user to provide input to, view output of, and invoke the functionality of the TDGM 204 and the TP 206. For example, a user may use UI 202 to invoke the TDGM 204 and to specify document generation options thereto, and to view and edit the transformation document 212 that is generated by the TDGM 204. Similarly, the user may use UI 202 to invoke the TP 206, and to view the result document 216 that is generated by the TP 206.

In response to an invocation from a user via the UI 202, the TDGM 204 generates a transformation document 212. In one embodiment, the TDGM 204 generates the transformation document 212 based upon a source document 208 and a target document 210. In alternative embodiments, the TDGM 204 may generate the transformation document 212 based solely upon the source document 208 or the target document 210. In addition, the TDGM 204 may generate the transformation document 212 based upon multiple source documents and/or multiple target documents. These and other implementations are within the scope of the present invention.

In generating the transformation document 212, the TDGM 204 in one embodiment attempts to create a document that, when processed with the source document 208, will produce the target document 210. Ideally, the transformation document 212 generated by the TDGM 204 is one that, when processed with the source document 208, will give rise to an exact replica of the target document 210. However, this is often not possible. In many instances, the TDGM 204 generates a transformation document 212 that, when processed with the source document 208, gives rise to an approximation of the target document 210. Given this transformation document 212, the user can make further edits using the UI 202 to refine the document 212 so that when the document 212 is processed with the source document 208, the result document is as close as possible to the target document 210.

Once generated, the transformation document 212 may be processed by the TP 206 in conjunction with a source document 214 to derive a result document 216. Source document 214 may be the same document as source document 208, or it may be another document of the same type as source document 208. One point to note regarding transformation document 212 is that it can be used to transform not just the source document 208 from which it was derived but rather any number of documents that are of the same type as source document 208. As a result, once created and tuned by the user, the transformation document 212 may be used to transform an entire class or batch of source documents 214. Based upon the source document 214 and the transformation document 212, the TP 206 produces a result document 216, which represents the transformed version of the source document 214, transformed in accordance with the transformation document 212. If the source document 214 is the same document as

source document 208, then the result document 216 will be at least an approximation of target document 210.

BACKGROUND INFORMATION

From a conceptual standpoint, the present invention as described herein may be applied to any type of source, target, and transformation document written in any language. To facilitate a complete understanding, however, the invention will be described below with reference to a specific example. In the following example, it will be assumed that the source document 208 and the target document 210 are XML documents, and that the transformation document 212 is an XSLT document. It should be noted, however, that this example is used for illustrative purposes only, and that it should not be construed to limit the invention in any way.

XML Source and Target Documents

In one embodiment, the source document 208 and the target document 210 are XML documents, and like most XML documents, take the form of text documents comprising one or more element tags and one or more data sets. As shown in the sample XML document of Fig. 1, the element tags may be nested within other element tags to give rise to a hierarchical structure, which defines the structural relationships between the various elements and sets of data. Because they specify a hierarchical structure, XML documents lend themselves to being represented by tree-type representations. In fact, many XML documents are not processed directly but rather are first parsed and transformed into tree representations, and then processed using the tree representations.

An XML document may be represented using any type of tree structure, but one tree structure that is commonly used is the one provided by the document object model (DOM). More specifically, an XML document may be parsed into objects using the DOM, and the DOM represents the parsed document as an object tree with a plurality of nodes. The DOM provides a rich tree representation for the XML document. Given any node on the tree, the DOM tree representation provides all information pertinent to that node. For example, the DOM tree provides information as to which node is the parent of that node, which nodes are children of that node, and which nodes are siblings of that node. Given this information, it can be easily determined where a particular node fits within the XML document. This is a very brief description of the DOM. More information and a specification for the DOM can be found on the W3C website at www.w3c.org.

XSLT Transformation Document

In one embodiment, the transformation document 212 takes the form of a document written in the XSLT language. In large part, an XSLT document comprises one or more templates. Each template generally comprises two parts: (1) a triggering pattern specification; and (2) one or more operations or actions. The triggering pattern specifies what structural pattern in a source document will cause the template to be triggered. Recall that the transformation document 212 is processed by the TP 206 in conjunction with a source document 214. If the source document 214 has the structural pattern specified in the triggering pattern of a template, then the TP 206 will invoke that template when the triggering pattern is encountered in the source document 214.

When a template is invoked, the operations or actions specified in that template are performed by the TP 206. According to XSLT, a number of different operations may be specified in a template. These operations include but are not limited to: (1) outputting a literal; (2) copying a pattern; and (3) applying templates. The "outputting a literal" operation causes the TP 206 to write a literal to the result document 216. The literal may be an element tag, an element value, an attribute value, or any other set of text. For example, suppose that it is desired to convert a "person" element tag in the source document 214 to an "employee" element tag in the result document 216. In such a case, a template may be created having "person" declared as the triggering pattern, and the action being to write the literals "<employee>" and </employee>" to the result document 216. That way, when the "person" pattern is encountered in the source document 214, the template is invoked and the literals "<employee>" and </employee>" are written to the result document 216.

The copying operation causes the TP 206 to copy an element and all of its child elements and data values directly from the source document 214 to the result document 216. This operation is useful when it is desirable to create in the result document 216 an identical copy of an element found in the source document 214. By identical, it is meant that the structural pattern and the data values within the element are the same. The copying operation will be elaborated upon in a later section.

The "apply templates" operation causes the TP 206 to apply all of the matching templates in the transformation document 212 to all of the children of a particular element in the source document 214. The apply templates operation is useful for fully processing

all of the children of a particular node in the source document. Use of the apply templates operation will be elaborated upon in a later section.

The above is a very brief description of the types of operations and actions that can be specified in an XSLT template. More information and a specification for XSLT can be found on the W3C website at www.w3c.org.

TDGM OPERATION

With the above background information in mind, the operation of one embodiment of the TDGM 204 will now be described. In the following description, reference will be made to some sample documents to fully illustrate the operation of the TDGM 204. Specifically, the XML document shown in Fig. 3 will be used as the sample source document 208, while the XML document shown in Fig. 4 will be used as the sample target document 210. The transformation document 212 will be generated based upon these sample documents. Operation of the TDGM 204 will be described with reference to the operational flow diagram shown in Fig. 5.

Initially, the TDGM 204 operates by receiving (502) a request from a user, via the UI 202, to generate a transformation document. Included in this request may be several sets of information, including but not limited to, information indicating which documents are to be the source and target documents, and information specifying the options, if any, according to which the transformation document 212 is to be generated. The options offered by the TDGM 204 may differ from implementation to implementation. For purposes of illustration, it will be assumed that the request specifies the documents shown in Figs. 3 and 4 as the source and target documents, respectively.

After receiving the document generation request, the TDGM 204, in one embodiment, proceeds to derive (506) a tree structure representation for each of the sample documents. In one embodiment, the TDGM 204 derives a tree representation by parsing a document in accordance with the DOM specification to give rise to an object tree. In an alternative embodiment, the TDGM 204 derives a tree representation by accessing an already parsed version of the document. Whichever is the case, the parsing of an XML document and the development of a tree representation according to the DOM is well known; thus, it need not be discussed in detail herein. With reference to Figs. 6 and 7, a tree representation for each of the sample documents is shown. Specifically, Fig. 6 illustrates a tree representation 602 for the sample source document 208 of Fig. 3, while Fig. 7 shows a tree representation 702 for the sample target document 210 of Fig. 4.

After the tree representations of the sample documents are derived, the TDGM 204 proceeds to analyze (510) each tree representation and to generate a pattern dictionary for each sample document to record all of the patterns that occur in that document. In generating a pattern dictionary for a document, the TDGM 204 in one embodiment traverses the tree representation for that document. Starting at the root node, the TDGM 204 traverses each node of the tree. For each node encountered, the TDGM 204 determines whether that node is a newly encountered node (i.e. whether the node already exists in the pattern dictionary). If the node does not already exist in the pattern dictionary, then the TDGM 204 adds that node to the pattern dictionary as a new pattern. Along with the node, the TDGM 204 stores a reference to where that node is located in the tree representation. This reference enables the TDGM 204 to quickly access the node on the tree at a later time. In this manner, the node is recorded in the pattern dictionary.

Suppose, however, that the node is not a newly encountered node but rather is one that already exists in the pattern dictionary. In such a case, the node is not inserted into the pattern dictionary as a new pattern. Instead, a reference to the node is just added to the existing node entry in the pattern dictionary. That way, the node is recorded as a reoccurrence of an existing pattern. For example, if there are two occurrences of a "person" pattern in a document, the pattern dictionary for that document would contain only one entry for the "person" pattern, but that entry would contain two references to the tree representation for that document. Each reference would refer to a particular location in the tree representation where the occurrence of the "person" pattern can be found.

If the tree representations 602, 702 of Figs. 6 and 7 are processed in the manner just described, the pattern dictionaries shown in Figs. 8 and 9 may be derived. Fig. 8 shows the pattern dictionary 802 derived for the sample source document 208 of Fig. 3, while Fig. 9 depicts the pattern dictionary 902 derived for the sample target document 210 of Fig. 4. Notice that each pattern dictionary 808, 902 comprises a complete list of all of the unique element nodes in the corresponding tree representation. Also notice that each entry of each pattern dictionary has an associated reference array. It is this reference array that stores a reference to each occurrence of the pattern in the entry in a corresponding tree representation. For example, for the "SourceDoc" entry in the pattern dictionary of Fig. 8, the reference array contains a reference to each location in the source document's tree representation (Fig. 6) where an occurrence of the "SourceDoc" pattern can be found. Because the pattern dictionaries include these references to the corresponding tree representations, the pattern dictionaries may be used to quickly access any occurrence of any pattern on any tree representation.

After the source and target documents 208, 210 are analyzed and the pattern dictionaries 802, 902 are built, the TDGM 204 proceeds to generate (514) the transformation document 212. In one embodiment, the document generation operation (514) comprises several parts: (1) generating the basic structure of the transformation document 212, which may include generating zero or more processing instructions (530); (2) generating pattern creation templates (534); and (3) generating copy templates (538). To illustrate how each of these parts is carried out, reference will be made to Figs. 10A-10D, which show a sample transformation document 212. This transformation document 212 is generated in accordance with one embodiment of the present invention based upon the sample source document 208 and the sample target document 210.

In generating (530) the basic structure of the transformation document 212, the TDGM 204 generates and inserts some basic information into the transformation document 212. As shown in portion 1004 of Fig. 10A, this information may comprise an indication that the document 212 is a transformation document, and a specification of where a namespace for the document is located. The basic information may also include zero or more processing instructions. These processing instructions may indicate, for example, where the source document 208 and the target document 210 may be found in a file system. The processing instructions may also indicate any options that were implemented to generate the document. These and other sets of information may be specified by the processing instructions. When the TP 206 processes the transformation document 212 in conjunction with a source document, the TP 206 may use the information in the processing instructions to determine one or more of its behaviors.

In addition to the basic information already described, the TDGM 204 also creates a basic template 1008 in the transformation document 212, the purpose of which is to start the processing of the transformation document 212 by the TP 206. As shown in Fig. 10A, template 1008 has a triggering pattern of "/". This means that it is triggered whenever the root node of the source document is encountered by the TP 206. Unless there is an error, the root node of the source document should be encountered every time the transformation document 212 is processed with a source document. Thus, template 1008 should be invoked every time.

When invoked, the template 1008 causes several actions to be performed. First, it causes the literal "<TargetDoc>" to be outputted to a result document 216 (note that TargetDoc is the name of the root node of the target document 210). Then, it causes all of the templates in the transformation document 212 to be applied to the children of the root node of the source document (this is the effect of the "xsl:apply-templates" action). Note that when the templates of the transformation document 212 are applied to the children of the root node, the templates will be triggered only if the children of the root node have the triggering patterns specified for the templates. After all of the templates have been applied to the children of the root node, the template 1008 outputs the literal "</TargetDoc>" to the result document 216. At that point, execution of the template 1008 is complete. Basically, the function of the basic template 1008 is to create a main element tag of "TargetDoc" in the result document, and to start document processing at the root node of the source document. With the template 1008 and the basic information of portion 1004 thus created, the foundation of the transformation document 212 is established.

After the document foundation is established, the TDGM 204 proceeds to generate (534) the pattern creation templates of the transformation document 212. The purpose of these templates is to ensure that when the transformation document 212 is processed with a source document, all of the patterns in the target document 210 are created in the result document 216. In one embodiment, each template causes one pattern to be created in the result document 216. According to one embodiment, the TDGM 204 generates the pattern creation templates by scanning through the pattern dictionary 902 (Fig. 9) of the target document 210, and creating a template for each pattern found in the pattern dictionary 902 (except for the root pattern TargetDoc, for which a template has already been generated). In one embodiment, a pattern creation template is generated as follows.

Initially, the TDGM 204 selects a pattern (e.g. "person") from the target document's pattern dictionary 902 (Fig. 9), and accesses the reference array for that pattern. Using a reference in the reference array, the TDGM 204 accesses a particular node on the tree representation 702 (Fig. 7) of the target document 210. This particular node is a node at which the pattern is found in the tree representation 702. Once the particular node on the tree representation is accessed, the TDGM 204 determines whether that node has any child nodes. Recall that the DOM tree representation provides a significant amount of information about a node, including whether the node has any child nodes. Thus, once the TDGM 204 accesses the particular node, the TDGM 204 can determine whether the particular node has any child nodes. Armed with the name of the

pattern and the knowledge of whether the pattern has any children, the TDGM 204 proceeds to generate the pattern creation template for the pattern.

To generate the template, the TDGM 204 first generates a general template structure. Within this structure, the TDGM 204 specifies a triggering pattern and a list of one or more actions. As noted previously, the triggering pattern dictates when the template is invoked, and the list of actions determines what the TP 206 will do when the template is invoked. Without purely guessing, it is difficult for the TDGM 204 to determine when a template should be invoked to create a pattern in the result document 216. Thus, in one embodiment, the TDGM 204 does not specify an actual triggering pattern, but rather sets the triggering pattern to "iis-pattern-needed". That way, when a user reviews the transformation document 212 after it has been generated, the user will know from the "iis-pattern-needed" indication that the user needs to provide a triggering pattern for the template.

After the triggering pattern is specified in the template, the TDGM 204 proceeds to specify the list of actions for the template. The list of actions specified for a template will depend upon whether the pattern being created by the template has children. If the pattern does not have children, then the TDGM 204 inserts one or more "output literal" operations into the template's action list. These output literal operations, when processed by the TP 206, will cause the TP 206 to create a particular pattern in the result document 216. For example, if the particular pattern for which the template is being created is the "person" pattern, then the actions of the template will comprise output literal operations for outputting the literals "<person>" and "</person>" to the result document 216.

If the pattern for which the template is being created has children, then in addition to the output literal operations noted above, the TDGM 204 further inserts an "apply templates" operation into the template action list. This will cause all of the matching templates of the transformation document 212 to be applied to all of the children of a particular node in the source document.

By applying the template generation process described above, the TDGM 204 generates the template 1012 shown in Fig. 10A for the "person" pattern of the target document's pattern dictionary 902. By applying the same process to each of the other patterns in the target document's pattern dictionary 902, the TDGM 204 generates all of the pattern creation templates 1014-1056 shown in Figs. 10B and 10C.

After the pattern creation templates are generated in the transformation document 212, the TDGM 204 proceeds to generate (538) zero or more copy templates. In generating the copy templates, the TDGM 204 initially determines whether there are any elements that are identical between the source document 208 and the target document 210. To be identical, two elements need to have identical structure and data values. If any identical element is found, then a copy template is generated in the transformation document 212 for that element. When the TP 206 processes the transformation document 212 in conjunction with a source document, this copy template will cause the TP 206 to copy the element from the source document to the result document 216.

In one embodiment, the TDGM 204 searches for identical elements between the source document 208 and the target document 210 in the following manner. The TDGM 204 initially selects one of the element entries in the source document's pattern dictionary

802 (Fig. 8). The TDGM 204 compares this element against all of the elements in the target document's pattern dictionary 902 (Fig. 9). If no match is found, then the TDGM 204 proceeds to the next element entry in the source document's pattern dictionary 802, and repeats the above process. On the other hand, if a matching element is found in the target document's pattern dictionary 902, then the TDGM 204 proceeds to determine whether the matching element is an exact match. In one embodiment, the TDGM 204 makes this determination by accessing and traversing the tree representations of the source and target documents.

To illustrate how this is done, reference will be made to an example. As shown in Figs. 8 and 9, there is a match in the pattern dictionaries 802, 902 for the "person" element. Thus, when processing the dictionaries 802, 902, the TDGM 204 will find this element match, and will try to determine whether the match is an exact match. To determine whether the match is an exact match, the TDGM 204 accesses the reference array associated with the "person" entry of the source document's pattern dictionary 802. From this array, the TDGM 204 obtains a reference. This reference points to a node on the source document's tree representation 602 (Fig. 6) where an occurrence of the "person" element can be found. Using this reference, the TDGM 204 accesses the appropriate node on that tree 602. Likewise, the TDGM 204 accesses the reference array associated with the "person" entry of the target document's pattern dictionary 902. From this array, the TDGM 204 obtains a reference. This reference points to a node on the target document's tree representation 702 (Fig. 7) where an occurrence of the "person" element can be found. Using this reference, the TDGM 204 accesses the appropriate

node on that tree 702. Once the tree representations 602, 702 are accessed, the TDGM 204 traverses the trees to determine whether the elements match exactly.

In one embodiment, the TDGM 204 performs the traversal by initially determining the children of the accessed nodes. As shown in Fig. 6, the "person" node of the source document has the nodes Name and Address as its child nodes. As shown in Fig. 7, the "person" node of the target document has the nodes Name and Residence as its child nodes. After the child nodes are determined, the TDGM 204 compares the child nodes to determine whether they are identical. If they are not (as is the case in the present example), then it is concluded that the element being tested (the "person" element) is not an exact match. In such a case, the TDGM 204 forgoes generating a copy template for the element, and proceeds to the next element in the source document's pattern dictionary 802 to look for an exact match for that element.

On the other hand, if the child nodes are identical, then the TDGM 204 proceeds further down the trees 602, 702 to test the child nodes of the child nodes. If all of those child nodes are identical, then the TDGM 204 further traverses the trees 602, 702 to test the child nodes of those child nodes. This process repeats until either a difference is found between the two elements, in which case the TDGM 204 concludes that the elements do not constitute an exact match, or all of the child nodes and data values have been tested and determined to be identical. If the elements are determined to be identical, then the TDGM 204 generates a copy template to copy the element from the source document to the result document 216.

In the sample source and target documents, an example of a matching element is the "pet" element. As can be seen from the tree representations shown in Figs. 6 and 7,

the "pet" element in both the source document and the target document have the two child nodes: Type and PetName. In addition, all of the corresponding child nodes have identical data values: "Cat" and "Tuffy". Thus, the "pet" elements match exactly. As a result, when the TDGM 204 processes the "pet" element in the source document's pattern dictionary 802, it will find an exact match, and hence, will generate a copy template for that element in the transformation document 212.

In one embodiment, the TDGM 204 generates a copy template by first generating a general template structure. Then, within this structure, the TDGM 204 specifies a triggering pattern. For a copy template, the triggering pattern is the element in the source document 208 for which an exact match was found in the target document 210. In one embodiment, the triggering pattern is specified in detail, indicating the full path to the element, and a specific instance of the element. For example, the triggering pattern for the "pet" element would be "/SourceDoc/pet[1]", where "/SourceDoc/Pet" indicates the full path to the element in the source document 208, and "[1]" indicates the first occurrence of the element in the source document 208.

In addition to the triggering pattern, the TDGM 204 further specifies in the template structure one or more template operations or actions. In one embodiment, for a copy template, the TDGM 204 inserts a single copy operation into the template. When invoked, this copy operation will cause the element specified in the triggering pattern to be copied to the result document 216. After the copy operation is inserted into the template structure, the generation of the copy template is completed. A sample copy template for the "pet" element is shown in Fig. 10C as template 1060. In the manner described, the TDGM 204 generates a copy template for each element in the source

document's pattern dictionary 802 for which an exact match is found in the target document 210. For the sample source document 208 and target document 210, the copy templates that are generated by the TDGM 204 are shown in Figs. 10C and 10D as templates 1060-1084. In the manner described, the TDGM 204 automatically generates the transformation document 212.

After the transformation document 212 is generated by the TDGM 204, the user may use the UI 202 to refine the transformation document 212. For example, the user may specify triggering patterns for the pattern creation templates. The user may also choose to delete some templates if they prove to be redundant or superfluous. In addition, the user may view the tree representations and the pattern dictionaries to further analyze the source and target documents. Overall, the user may refine the transformation document 212 in any way to achieve improved/desired results.

HARDWARE OVERVIEW

In one embodiment, the various components 202, 204, 206 of the present invention are implemented as sets of instructions executable by one or more processors. The invention may be implemented as part of an object oriented programming system, including but not limited to the JAVA™ programming system manufactured by Sun Microsystems, Inc. of Palo Alto, California. Fig. 11 shows a hardware block diagram of a computer system 1100 in which an embodiment of the invention may be implemented. Computer system 1100 includes a bus 1102 or other communication mechanism for communicating information, and a processor 1104 coupled with bus 1102 for processing

information. Computer system 1100 also includes a main memory 1106, such as a random access memory (RAM) or other dynamic storage device, coupled to bus 1102 for storing information and instructions to be executed by processor 1104. Main memory 1106 may also be further used to store temporary variables or other intermediate information during execution of instructions by processor 1104. Computer system 1100 further includes a read only memory (ROM) 1108 or other static storage device coupled to bus 1102 for storing static information and instructions for processor 1104. A storage device 1110, such as a magnetic disk or optical disk, is provided and coupled to bus 1102 for storing information and instructions.

Computer system 1100 may be coupled via bus 1102 to a display 1112, such as a cathode ray tube (CRT), for displaying information to a computer user. An input device 1114, including alphanumeric and other keys, is coupled to bus 1102 for communicating information and command selections to processor 1104. Another type of user input device is cursor control 1116, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 1104 and for controlling cursor movement on display 1112. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane.

According to one embodiment, the functionality of the present invention is provided by computer system 1100 in response to processor 1104 executing one or more sequences of one or more instructions contained in main memory 1106. Such instructions may be read into main memory 1106 from another computer-readable medium, such as storage device 1110. Execution of the sequences of instructions

contained in main memory 1106 causes processor 1104 to perform the process steps described herein. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement the invention. Thus, embodiments of the invention are not limited to any specific combination of hardware circuitry and software.

The term "computer-readable medium" as used herein refers to any medium that participates in providing instructions to processor 1104 for execution. Such a medium may take many forms, including but not limited to, non-volatile media, volatile media, and transmission media. Non-volatile media includes, for example, optical or magnetic disks, such as storage device 1110. Volatile media includes dynamic memory, such as main memory 1106. Transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus 1102. Transmission media can also take the form of acoustic or electromagnetic waves, such as those generated during radio-wave, infra-red, and optical data communications.

Common forms of computer-readable media include, for example, a floppy disk, a flexible disk, hard disk, magnetic tape, or any other magnetic medium, a CD-ROM, any other optical medium, punchcards, papertape, any other physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, any other memory chip or cartridge, a carrier wave as described hereinafter, or any other medium from which a computer can read.

Various forms of computer readable media may be involved in carrying one or more sequences of one or more instructions to processor 1104 for execution. For example, the instructions may initially be carried on a magnetic disk of a remote

computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system 1100 can receive the data on the telephone line and use an infra-red transmitter to convert the data to an infra-red signal. An infra-red detector can receive the data carried in the infra-red signal and appropriate circuitry can place the data on bus 1102. Bus 1102 carries the data to main memory 1106, from which processor 1104 retrieves and executes the instructions. The instructions received by main memory 1106 may optionally be stored on storage device 1110 either before or after execution by processor 1104.

Computer system 1100 also includes a communication interface 1118 coupled to bus 1102. Communication interface 1118 provides a two-way data communication coupling to a network link 1120 that is connected to a local network 1122. For example, communication interface 1118 may be an integrated services digital network (ISDN) card or a modem to provide a data communication connection to a corresponding type of telephone line. As another example, communication interface 1118 may be a local area network (LAN) card to provide a data communication connection to a compatible LAN. Wireless links may also be implemented. In any such implementation, communication interface 1118 sends and receives electrical, electromagnetic or optical signals that carry digital data streams representing various types of information.

Network link 1120 typically provides data communication through one or more networks to other data devices. For example, network link 1120 may provide a connection through local network 1122 to a host computer 1124 or to data equipment operated by an Internet Service Provider (ISP) 1126. ISP 1126 in turn provides data communication services through the world wide packet data communication network now commonly

referred to as the "Internet" 1128. Local network 1122 and Internet 1128 both use electrical, electromagnetic or optical signals that carry digital data streams. The signals through the various networks and the signals on network link 1120 and through communication interface 1118, which carry the digital data to and from computer system 1100, are exemplary forms of carrier waves transporting the information.

Computer system 1100 can send messages and receive data, including program code, through the network(s), network link 1120 and communication interface 1118. In the Internet example, a server 1130 might transmit a requested code for an application program through Internet 1128, ISP 1126, local network 1122 and communication interface 1118. The received code may be executed by processor 1104 as it is received, and/or stored in storage device 1110, or other non-volatile storage for later execution. In this manner, computer system 1100 may obtain application code in the form of a carrier wave.

At this point, it should be noted that although the invention has been described with reference to a specific embodiment, it should not be construed to be so limited. Various modifications may be made by those of ordinary skill in the art with the benefit of this disclosure without departing from the spirit of the invention. Thus, the invention should not be limited by the specific embodiments used to illustrate it but only by the scope of the appended claims.